1998-03-01

An approach to mobile agent security in Java

Virden, Roy John

Monterey, California. Naval Postgraduate School

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN APPROACH TO
MOBILE AGENT SECURITY IN JAVA

by

Roy John Virden

March, 1998

Thesis Advisor: Dennis M. Volpano
Second Reader: Nelson D. Ludlow

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
AN APPROACH TO MOBILE AGENT SECURITY IN JAVA

5. FUNDING NUMBERS

6. AUTHOR(S)
Virden, Roy John

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

For many years, people have talked about the advantages of programs that can roam networks and provide services for a client. The programs, called agents, have many military applications as well. Among them, for instance, is data mining, where an agent is dispatched to find information for a client.

There are security risks associated with agents. For instance, in the data mining example, a client must be able to trust the information returned. If a trusted node in a network can be spoofed, then an untrusted node can easily corrupt the results of the mining operation.

This thesis presents a protocol to guard against this sort of attack. The protocol assumes that every trusted host knows all other trusted hosts. Though unrealistic for some commercial applications, it seems like a reasonable assumption for military applications.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Mobile Agent, Security, Java, Authentication protocol	99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

AN APPROACH TO MOBILE AGENT SECURITY IN JAVA

Roy John Virden
Lieutenant, United States Navy
B.A., Miami University, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1998

ABSTRACT

For many years, people have talked about the advantages of programs that can roam networks and provide services for a client. The programs, called agents, have many military applications as well. Among them, for instance, is data mining, where an agent is dispatched to find information for a client.

There are security risks associated with agents. For instance, in the data mining example, a client must be able to trust the information returned. If a trusted node in a network can be spoofed, then an untrusted node can easily corrupt the results of the mining operation.

This thesis presents a protocol to guard against this sort of attack. The protocol assumes that every trusted host knows all other trusted hosts. Though unrealistic for some commercial applications, it seems like a reasonable assumption for military applications.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Many thanks to my patient and loving wife, Susan, who has been a great source of strength and encouragement all through this work

Thanks also to Professor Volpano and Major Ludlow, for letting me proceed on my own until I needed some guidance, and for being right there when asked.

Thanks to Professor Irvine, Professor Hensgen and the entire NPS CISR department for all the support and confidence in my ability.

I. INTRODUCTION

A. BACKGROUND

Webster's New World Dictionary defines an *agent* as "a person or thing that performs an action or brings about a certain result, or that is able to do so." The term agent has been used for many years in computer science to describe a program that performs a job on a user's behalf. An agent normally executes on a single machine and performs tasks of varying complexity and ability. One use of agents can be found in today's networks and distributed processing systems. "Key platforms, such as hosts, bridges, routers, and hubs, may be equipped with agent software so that they may be managed from a management station (Stallings, 1997)." The agent, in this case, monitors the node's active status and responds to information requests from network management stations. Spelling checkers integrated in modern word processors can also be considered agents. They simply monitor or scan input text, reporting any differences between input strings and their own dictionaries to the main word processing program.

Software agents are rooted mostly in the artificial intelligence and distributed systems areas of computer science. The availability of digital computers and the assumption that real world aspects can be symbolically represented gave rise to the research area of artificial intelligence at the Dartmouth Conference in 1956. The development of agent research has been influenced by five parent disciplines: control theory, cognitive psychology, artificial intelligence planning theory, object-oriented programming and distributed systems. (Muller, 1996)

Control theory investigates the agent-world relationship from a machine-oriented perspective whereas cognitive psychology deals with behavior and motivation theory. In

the 1970's, classical artificial intelligence planning systems strongly influenced agent design, viewing the problem-solving behavior of agents as a sense-plan-act cycle. The 1980's brought the notion of the agent playing a central role in the research of distributed artificial intelligence. (Muller, 1996)

Agents are historically divided into three classes: deliberative, reactive and interacting. Deliberative agents rely on an internal representation of their world, and base their actions on some form of complex symbolic reasoning. These agents are usually modeled on beliefs, desires and intentions. In the mid 1980s, a new school of thought emerged that was influenced by behaviorist psychology. It led to reactive agents. These agents make their decisions at run-time, based on environmental sensor input. They contain limited amounts of internal representation. The interacting agent class, beginning in the late 1980s, focused on the coordination process and on mechanisms for cooperation among autonomous agents rather than on the structure of these agents. In the past decade, a considerable amount of effort has been dedicated to combining these classes in order to overcome their individual limitations. (Muller, 1996) Blackboard Architecture is one example.

Also in the 1980s, Minsky's views became prominent in the world of agents and intelligence. Minsky distinguishes between agents and agencies. In Minsky's *Society of Mind*, an agent is a simple non-intelligent part or process. On the other hand, an agency is a collection or society of these simple agents demonstrating the appearance of intelligence. (Minsky, 1985)

With the advent of Java, there is renewed interest in the mobility of agents. The Java Virtual Machine and Java's class loading model, coupled with serialization, remote

method invocation and multithreading, have made prototyping mobile agent systems a fairly straightforward task.

Mobile is defined as "moving, or capable of moving or being moved, from place to place." Thus, a mobile agent can perform varying tasks and can move from host to host throughout a network. It has the ability to decide if and when it needs to move and then request to be transported to the desired location. While traveling through a network, it can search for information and execute commands at a remote server, eventually reporting back to its client when its task is complete (Kalakota, 1996).

1. Mobile Agents in the Military

Software agents offer tremendous potential in supporting the Department of Defense. Military intelligence analysts can benefit from a mobile agent that provides remote sensor observation, data collection and situation reporting. Perhaps the greatest benefit to the analyst is the dynamic nature of the agent. In a high operational tempo scenario with continually changing situations and requirements, an analyst can dispatch an agent with a request for information based on the most recent developments. An agent can also encode decision-making logic in order to make decisions while at remote locations.

Another benefit to the military is the ability for a soldier in the field with a handheld device to dispatch an agent to a command and control center requesting further instructions or local area intelligence data. The soldier can then shut down the device, perform evasive ground maneuvers and then, at a later time, restart the device in order to retrieve the results of the request.

Security is important to military users of mobile agents. The intelligence analyst needs precise information when providing Indications and Warning support to forward-deployed units and the soldier has to rely on his marching orders to stay out of harm's way. Thus, an agent's client must have the confidence and trust that a dispatched agent will execute in the manner desired and that all collected data are not corrupt.

B. PROBLEM STATEMENT

The objective of this thesis is to develop a trusted mobile agent model permitting a client to dispatch an autonomous agent into a network of databases and upon return of the agent, have confidence that the agent has not been subverted.

C. APPROACH

The approach is based on a trusted mobile agent model that uses a host-to-host authentication protocol and public-key cryptography. Agents originate from trusted hosts and are forwarded only to other trusted hosts. Every trusted host has a list of hosts which it trusts. Certificates issued by a central issuing authority are used in the authentication process. Mobile agents will be executed at remote hosts only following a successful authentication agreement. Upon an agent's return to a client, we know the agent has visited only trusted hosts. Therefore, any returned results can be used with confidence. We believe that the trusted mobile agent model can be useful in realizing mobile agent applications in the military.

D. THESIS ORGANIZATION

Chapter II presents software mobile agents, providing examples that demonstrate practical mobile agent use. Chapter III presents a trusted mobile agent model, discusses

the associated security risks and lists the simplifying assumptions. Chapter IV gives details of an implementation of the model in Java and explains mechanisms used for mobility and authentication. In Chapter V, a military related scenario is developed and discussed. Chapter VI provides a summary and conclusions.

II. MOBILE AGENTS

A. WHAT IS A MOBILE AGENT?

A mobile agent is considered autonomous and, in general, consists of executable program code, along with some form of execution state. It carries with it everything required to perform its tasks and need not rely on previously-visited hosts for execution. An agent can execute on a particular host and decide it needs to transfer to another host. It can then save its state, halt execution and forward itself to that host. Once it arrives at the new host, it may continue where it left off.

With movement as a characteristic, a mobile agent should maintain a sense of location or host identification. A mobile agent may have a home from which it originates and can dispatch itself to a remote location or locations and perform programmed operations. Eventually it may return home with a result or communicate results back to the client via email or some other data transmission protocol. The agent's journey may be predefined with a planned itinerary or destinations may be determined as it travels, depending on navigation decisions made at each stop.

B. WHY MOBILE AGENTS?

The benefits of mobile agents can be separated into two levels: a user level and a distributed-system level.

At the user level, agents, in general, improve productivity by reducing client workload, allowing more time for other activities. Kalakota and Whinston (Kalakota, 1996) list some typical reasons for software agents: managing information overload, decision support, repetitive office activity, mundane personal activity, search and

retrieval, and domain experts. They also state the most important tasks performed by an agent are "gathering information, filtering information and using it for decision making." These three benefits are valuable in a military setting.

An intelligent mobile agent can travel throughout a network and make real-time decisions, requiring no interaction with the sender. While trying to satisfy a query, it can decide where it should go to find the necessary data for computation. Once there, it can perform data filtering on behalf of the sender and, when necessary, may either return home with the result or transmit it via other means. Since no communications are required while in transit, the agent's client is free to perform other activities. This also permits the agent to respond more quickly to the client with the result of the query.

The distributed-system level provides even more justification for military mobile agent applications. Military databases often contain large amounts of information such as own-force status and location, cartography details, forecasted weather data, reconnaissance findings, collected intelligence data and logistical statistics. Operational planning requires a large mix of all these extremely dynamic databases, which are rarely centrally located. Sending an agent out to search through these databases can prove to be more efficient than maintaining a continuous connection, say with a database server. A continuous connection often is unnecessary.

Agents performing data mining may need fewer packets, depending on the type of filtering operation they perform. Consequently, distinguishing changes in the status of forces by monitoring network traffic becomes more difficult.

Mobile agents offer the flexibility of providing immediate notification upon finding a desired piece of information during a data mining operation. For example, an

agent could be programmed to search a list of hosts, collecting data along the way. If it collects certain information that is contained in a predefined high-priority set that it carries, then it may choose to transmit a response immediately back to the client.

The advent of mobile devices, such as laptops and personal, handheld communicators are served well by mobile agent technology. Mobile devices share the following three characteristics which demand the kind of support provided by mobile agents (Harrison, 1995):

- They are only intermittently connected to a network, hence have only intermittent access to a server.
- Even when connected, they have only relatively low-bandwidth connections.
- They have limited storage and processing capacity.

Soldiers on the move, as mentioned in Chapter I, or submarines on missions that allow them to surface only when necessary, could be users of intermittent connections provided by agents. A military unit can formulate its request for information and dispatch the agent via a short burst communications transmission. The unit could then shut down communication lines, continue with the local mission at hand and, at a later time, re-establish communications and retrieve the mobile agent's reply.

C. EXAMPLES

A popular example often used when describing commercial use of mobile agents is an airline reservation scenario. Versions of this scenario can be found in (Chess, 1995), (Farmer, 1996) and (Yee, 1997). Farmer focuses on four hosts: a customer host, a travel agency host and two servers owned by competing airlines (Farmer, 1996). Chess uses a similar scenario (Chess, 1995). Yee studies a mobile agent that travels through a series of airline reservation servers (Yee, 1997). In both cases, it can be assumed that a travel agency programs the agent to provide a service for a customer.

Generally, in these examples, a customer desires to make an airline flight reservation based on destination, flight time availability and lowest cost. An agent, acting on behalf of a client, visits a series of airline servers. At each server, the agent queries the flight database and stores the results. Having collected enough flight information, the agent decides on a travel plan based on the customer's desires. The agent may then forward itself to the airline server with the "best" offer, make the reservation and, finally, return home with the results.

There are a number of security concerns associated with this example based on the competitive nature of airline agencies in the commercial world. First, we must note that "it is impossible to hide anything within an agent without the use of cryptography (Chess, 1995)." Cryptography can conceal collected data but does not prevent a malicious host from corrupting or deleting it. Also, in order for the agent to process the collected data it must be readable. This means it must not be encrypted or the agent must carry the decrypting key. Now since it is impossible to hide anything within an agent, the latter provides the malicious host the opportunity to obtain the key and read collected data. Therefore all hosts will have access to data contained within an agent's state.

In the airline flight reservation example, a malicious airline server could attempt to win business by altering flight records collected so far. The greedy server might raise its competitor's fares in hopes of being chosen as the airline with the lowest cost or just delete the other records altogether. In these competitive commercial systems, secrecy and integrity are important.

Another practical use of mobile agents is providing safeguards and counter-measures in distributed systems. The safeguard of intrusion detection in a network can

greatly enhance system security. Farmer describes an intrusion protection system of mobile agents that actively monitor network activity for suspected attacks. Once an intrusion has been detected, a response team of agents is deployed to activate the appropriate counter-measures. (Farmer, 1996) In the case of a network virus being detected, an agent could be sent out transporting a bug fix along with instructions for applying it (Black, 1997). Employing agents in these distributed detection and response systems makes use of an agent's mobility, flexibility and decision-making capabilities.

There are also security issues involved in these distributed intrusion detection and response systems. As mentioned in (Farmer, 1996), intruders could debilitate a host system on which the agents need to run by manipulating the server in some way or just causing it to crash. An intruder might insert hostile agents or attempt to alter or trick the legitimate agents into performing malicious tasks. These issues point out the importance of agent authentication, integrity and availability.

Another example is procurement. Here multiple agents attempt to bid for goods or services offered by an auctioneer agent or host. This is considered a more complex model in (Chess, 1995). The agent's goals and monetary resources must be hidden from other agents and untrusted hosts. Typical procurement examples include electronic malls, flea markets and sealed bidding auctions. This example provides a challenging security exercise in attempting to devise a method ensuring that each agent's attributes remain secret.

Agents can also be useful in the graphics world. Minar presents an example where a computer animation is being constructed by a large entertainment firm (Minar, 1996). A mobile agent first visits a host that holds the requirements of the construction. The

agent then moves to a 'render farm' and spawns many agents to produce the frames of the animation. Finally, the agent collects the frames and takes them to a final production host that combines the frames and packages the resulting movie. Depending on the proprietary concerns of the firm's product, the secrecy and integrity requirements in this example can be critical.

A mobile agent's dynamic nature allows it to actively respond to real-time events in the fast-paced, changing world of distributed systems. Whether employed in a commercial application where electronic monetary accounts are of prime concern or in a military setting where a nation's defense and human lives are at stake, mobile agent security needs to be addressed. This thesis addresses some of these issues in a military scenario, namely data mining, and develops an agent transmission protocol that permits use of mobile agents with confidence.

III. AN APPROACH TO MOBILE AGENT SECURITY

A. THE MOBILE AGENT MODEL

What is our basic mobile agent model? The basic mobile agent model contains the following components: a mobile agent, a client, a network of hosts and host databases.

A mobile agent, in general, consists of executable program code and some form of execution state. This model represents mobile agents as agent folders. An *agent folder* contains executable code, state information, a client's identification and password and perhaps a session key. All gathered data or dynamic state information is stored in the state field of the folder. The client's identification and password are provided in case client authentication is needed at remote hosts. If an agent is required to collect classified data then a session key is provided in order to encrypt the data while in transit. An *agent packet* is the basic container used to transport agent folders from host to host.

The agents can be programmed to gather information, fuse the collected data, eliminate redundancies, highlight conflicting information and, lastly, summarize the results, creating a useful product.

An agent executes at a host as long as it takes to complete its mission. If an agent has completed its mission and does not need to be forwarded to another host, then it terminates immediately.

An agent may be dispatched to another host for one of the following reasons:

- Initial dispatch from the client.
- The agent determines that the requested information is not available on the current host and forwards itself onward to another host listed in a pre-planned itinerary.

- The current host could recommend other hosts it should visit that may have relevant information.
- Dispatching could be triggered by the collection of a certain piece of information requiring the agent to return to the client for continued operations. For example, upon notification that a particular type of enemy aircraft has been launched from a particular airfield, the agent needs to go home to display the information and insert the data into the client's database.

An agent can forward itself, if necessary, to visit any number of hosts simultaneously. Further, an agent communicates only with hosts on which it resides. There is no agent-to-agent communication in this model for reasons of simplicity.

The client is a user on the originating host. This is where the mobile agent initially executes. The client dispatches an agent and the agent returns to the client with its results.

Network hosts are all interconnected machines, each providing similar socket connection services. Each network host has an *agent packet server* listening for a connection request to be received. Upon receipt, the executable code is run and the state information is made available for agent use.

Databases reside on each host. They are considered read-only databases with respect to visiting mobile agents. Agents are given only read access on all hosts, except the originating host. This is done only to keep the model simple. The agent can access the databases either directly or via a system specific interface. Agent execution may differ according to mission or application and will be configured accordingly to interact with the intended database or system interface. Agent programmers have advance knowledge of database system interfaces on each machine that may be visited by the agent.

B. SECURITY RISKS

Military operational environments often require requests for intelligence or targeting information. It is common for the results to be used in making decisions that could endanger human safety or affect delicate foreign relations. The basic model permits a host system to dispatch an autonomous agent throughout a network of databases, collect data and return a result. Associated with the basic model are a number of security concerns.

Security threats to a system fall into three aspects: secrecy, integrity and availability. Secrecy ensures that users only access information to which they are allowed. Integrity means a process remains free from corruption and unauthorized changes. Authentication verifies the origin of the sender. Availability means that the computer system's hardware and software keeps working efficiently and the system is able to recover quickly and completely if a disaster occurs. (Russell, 1991)

A mobile agent has two generalized locations. The agent is either resident on a host or in-transit between hosts. While an agent is executing on a host it is vulnerable to all three security threats. As noted in Chapter II, hosts have access to all data contained in an agent. Agent secrecy, integrity and availability are all assumed to rely on the host being well behaved.

An agent moving from host to host is susceptible to common network attacks. Communications media and equipment are points of vulnerability to any data being transmitted. A mobile agent can be monitored to obtain private information. The agent could even be altered with an unauthorized modification during network travel, resulting

in an integrity violation. Although denial of service attacks exist, they are beyond the scope of this thesis and are not considered.

These threats to mobile agent secrecy and integrity are treated by the following protocol.

C. SECURE MOBILE AGENT TRANSMISSION PROTOCOL

We begin with a base protocol we call forward-and-authenticate. It does not require a certification authority and guarantees secrecy, integrity and authentication of mobile agent folders while in-transit. Then we describe a variation of this protocol called authenticate-forward-authenticate. It is based on the Secure Password Transmission Protocol (Volpano, 1997) and requires a certification authority. The Secure Password Transmission Protocol provides the secure transmission of a password from a client to a server followed by the secure transmission of information from that server back to the client. The Secure Mobile Agent Transmission Protocol, however, provides authentication and safe transmission of mobile agents from host to host.

A public key cryptography system is used and it is assumed all hosts have access to the public key of the host to which they wish to forward an agent. It is also assumed that receiving hosts have access to the public key of the sender. See figure 1. When an agent executing on host A, requests to be dispatched to another host, B, the sending host encrypts the agent folder (af) with host B's public key, P_B, to provide secrecy. Host A then uses a mutually available one-way hash function, H, to produce a hash of the agent folder, H(af). This hash is encrypted with Host A's private key, S_A to produce $(H(af))^{S_A}$. The encrypted hash of the agent folder will provide authentication and integrity and simulates a digital signature. We use a single key pair for both asymmetric cryptography

and digital signatures. The resulting agent folder, hash and host's name are then sent to host B in the agent packet container (ap).

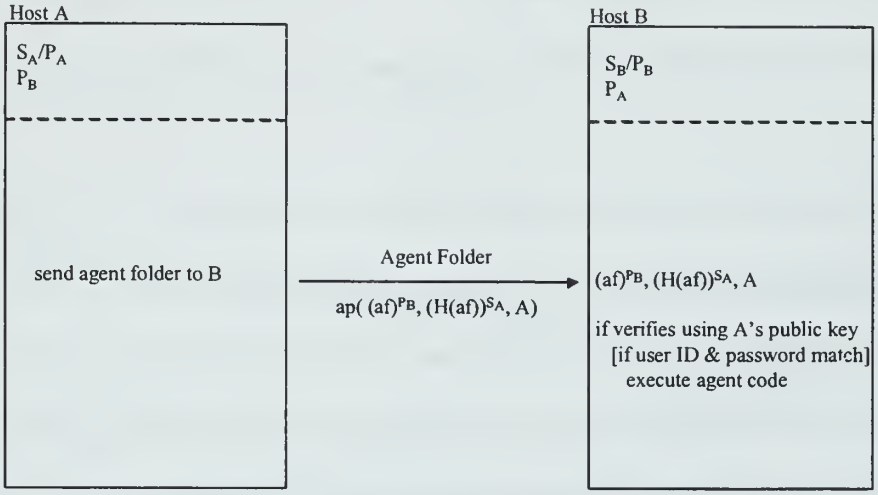

Figure 1. Basic Secure Mobile Agent Transmission Protocol

Upon receipt, host B decrypts the received agent folder with its private key, S_B, and uses the same hash function, H, to produce a hash of the received agent folder. The received hash is decrypted with the public key of host A, P_A. (If B has no public key for the host name provided in the folder, then it drops the folder.) If the two hashes match then the received agent folder is authenticated to be from host A and has not been modified. Host B then performs any necessary authentication of the client on whose behalf the agent wishes to execute. This will require the password in the agent folder. Finally B executes the agent.

The same protocol applies whether the agent requests to be forwarded to another host or to return home.

A variation is now introduced where the connecting hosts do not know each other's public keys. It is an authenticate-forward-authenticate protocol. Here a certificate-based system is used and it is assumed that all hosts have access to the public key, P_{CA}, of a central certification authority, CA. Certificates in this model, $CERT_A$ and $CERT_B$, include host name and public key only and are encrypted with the CA's private key, S_{CA}.

All connected hosts initially acquire the CA's public key, P_{CA}. Next, in order to receive mobile agent packages, a host obtains a signed certificate from the CA, $(CERT_A)^{S_{CA}}$, containing the host's identity and public key. At this point, the host is ready to send or receive mobile agent packages. See Figure 2. Agent packages also contain a boolean field indicating whether the agent packet is a certificate request or a packet containing the encrypted agent folder along with its hash and the host's certificate. Empty fields are represented by null in the figure.

When an agent requests to be dispatched, the sending host, A, connects to a remote receiving host B and requests a certificate for B. Host B replies by sending its certificate, $(CERT_B)^{S_{CA}}$, which is host B's name and public key, encrypted with the CA's private key.

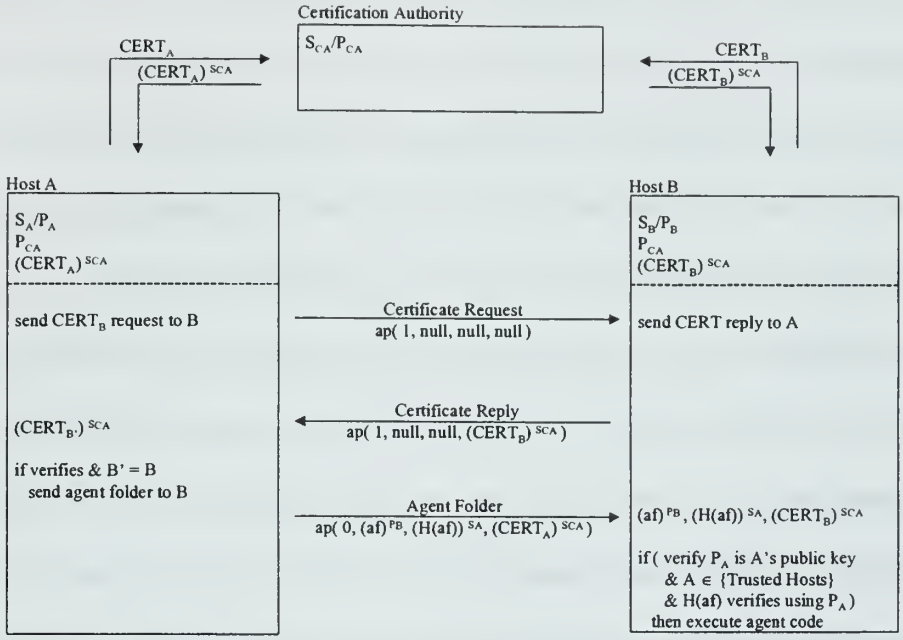

Figure 2. Secure Mobile Agent Transmission Protocol with Certificate Authority

Upon receipt, host A decrypts host B's certificate with the CA's public key revealing a host name and associated public key. This confirms that the public key in the certificate is the public key of the received host name found in that certificate. Host A compares the received host name B' with the requested host name B. If they match, host A now has host B's public key. Next, as in the original protocol, the agent folder is encrypted with host B's public key to provide secrecy, $(af)^{P_B}$. Host A then uses a mutually available one-way hash function, H, to produce a hash of the agent folder, H(af). This hash is encrypted with Host A's private key to produce $(H(af))^{S_A}$. The encrypted hash of the agent folder will provide authentication and integrity, and simulates a digital signature. Again, this method of signing, with the already available keys, allows the server to maintain only one set of keys per connected host instead of two. The

resulting agent folder, hash and Host A's certificate, $(CERT_A)^{S_{CA}}$, are then sent to host B in an agent packet.

Upon receipt, Host B decrypts the received certificate with the CA's public key revealing a host name and associated public key. If host B chooses to communicate with the received host name, it decrypts the received agent folder with its private key and uses the same hash function, H, to produce a hash of the received agent folder. The received hash is decrypted with the public key of host A. If the two hashes match then the received agent folder is authenticated to be from host A and has not been modified. Host B then performs any necessary authentication of the client, using the password provided in the agent folder, and then executes the agent.

D. SIMPLIFYING ASSUMPTIONS

It is important to note that certain simplifying assumptions have been made in developing a model demonstrating that mobile agents can be used in confidence, employing the standard uses of public key cryptography and certificates. The following assumptions make this possible.

In the military world, it is common to communicate with, and request information from, a predetermined set of suppliers in which some degree of trust exists. Examples would be national and theater intelligence collection and analysis centers, mapping agencies, meteorological and oceanographic centers, etc. This model uses this characteristic in that it maintains a list of trusted hosts. Thus, agents will only be forwarded to other trusted hosts.

All useful agents originate from a trusted host and are only forwarded to other trusted hosts. Any originating trusted host is the one from which an agent is launched

and to which it eventually reports the result of its execution. A trusted host promises to correctly execute agent instructions and not to violate the integrity of the software agent's content. Included is the promise not to misuse the secret session key included in the agent folder which is used by a host to encrypt sensitive data in an agent's folder.

Trusted hosts promise to keep their servers secure from external attackers using common available methods such as firewalls, virus scanners and strong identification and authorization mechanisms.

Each trusted host knows all other trusted hosts to which it is connected and which it trusts. A trusted host may be connected to other hosts that it considers not trusted or of unknown safety. A host will authenticate the public key of another trusted host on behalf of an agent prior to forwarding the agent to that host. A host will not attempt to transfer an agent to a host not considered trusted.

All trusted hosts consist of an agent handler daemon and associated utility software including a public key cryptography suite. All trusted hosts use the Secure Mobile Agent Transmission Protocol, described in this thesis.

A certification hierarchy will most likely be used in networks with a large number of trusted hosts. This model assumes a single CA is used. Certificates normally include a specific validation period and, upon expiration, new certificates are negotiated. This model assumes that certificates have unlimited lifetime and does not include timestamps in the certificates. Certificates sometimes use random numbers as one-time pads preventing replay. This model does not include generated random numbers in certificates. These methods and features treat important security issues. Certificates in this model include host name and public key only for reasons of simplicity.

These simplifying assumptions are essential to the success of this model. If an agent successfully returns to the original client then the result can be trusted. Here, trusted means the agent returns with results that can be obtained by visiting only trusted hosts in the network. The strength of this statement ultimately rests on the strength of public key cryptography and on the ability to keep private keys private.

IV. IMPLEMENTING MOBILE AGENT SECURITY IN JAVA

A. THE BASIC PROTOTYPE

The Java Virtual Machine and Java's class loading, coupled with serialization, networking, multi-threading and the cryptography architecture have made prototyping the trusted mobile agent model a fairly simple task. The prototype was developed on four networked Sun SPARC Station 10 machines with the Solaris V2.5 operating system using JDK1.1.

The mobile agent implementation is based on my advisor's active network design and much of the source code is rooted in the exercises and projects from the CS3973 Advanced Object-Oriented Programming in Java course at the Naval Postgraduate School, Monterey, California.

The implementation is a certificate-based authenticate-forward-authenticate protocol. The code differs slightly from the Secure Mobile Agent Transmission Protocol model in that Host A's certificate is forwarded in the initial request verses being forwarded in the final agent packet which contains the agent folder.

An agent packet is the basic container used to transport mobile agents from host to host. The agent packet contains an agent folder and a signed certificate using the Digital Signature Algorithm, DSA, supported by the cryptography architecture. A boolean flag indicates whether the packet is carrying an agent folder or a signed certificate. This flag is used in implementing the transport protocol. See Figure 3.

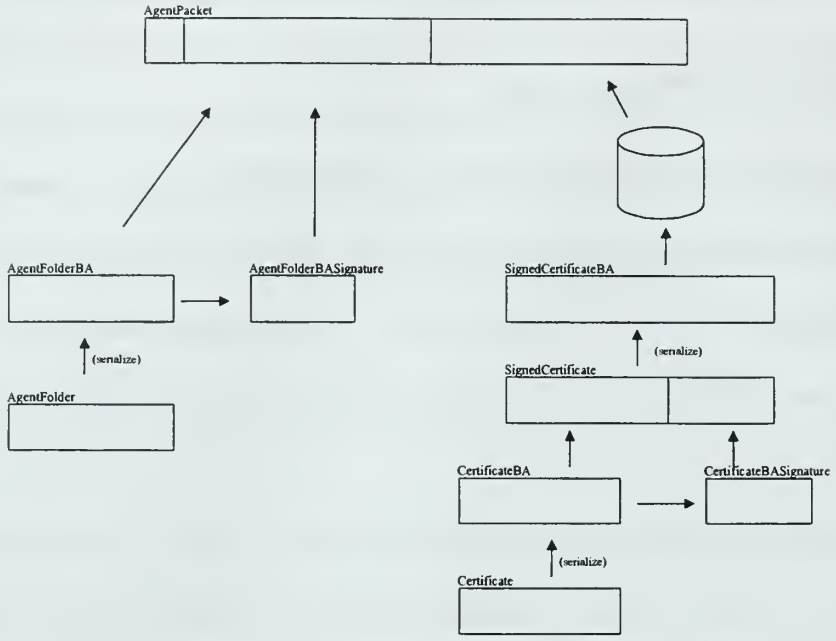

Figure 3. Agent Packet with DSA Design.

The agent folder contains five members: originating client's identification, user password, agent bytecode, agent state, a boolean value signifying if the session key is in use and a session key. A BA at the end of the variable name simply indicates the form of a byte array.

```
public class AgentFolder implements Serializable {
   public String  userID;
   public String  password;
   public byte[]  agentCodeBA;
   public byte[]  agentStateBA;
   public boolean encrypted = false;
   public byte[]  sessionKeyBA;
}
```

The basic mobile agent model implements a multi-threaded TCP server class called `AgentPacketServer`. This server listens for a connection request on port 6011 (arbitrary) for an agent packet to be received. Once the server receives an agent

packet, it creates a new thread and adds it to a `CPUscheduler`, which handles the scheduling of incoming agents.

The `AgentPacketServer` then organizes incoming agents into executable class files. It does this by instantiating the `agentCode` bytecode contained in the `AgentFolder`. The mobile agent's source code is contained in the `AgentCode` class that implements the `AgentCodeInterface`:

```
public interface AgentCodeInterface {
   public void exec(agentUtil.AgentFolder b,
                    agentUtil.AgentCodeUtilityInterface u)
                                        throws Exception;
}
```

The `AgentCode` class consists of the single method `exec`. This method contains the decision-making logic of the agent and is called by `AgentPacketServer`. The pseudocode for the military scenario can be seen in Figure 4.

```
if at Joint Task Force Headquarters
    if initial visit
        forward agent to Theater Intelligence Headquarters
    else
        display collected data

if at Theater Intelligence Headquarters
    search database for locations able to observe F-4 activity
    forward agent to Sub-Regional Intelligence Center

if at Sub-Regional Intelligence Center
    search database for data related to F-4 activity
    forward agent Joint Task Force Headquarters
```

Figure 4. Military Scenario pseudocode.

The `exec` method takes, as parameters, the incoming agent folder and an instance of the `AgentCodeUtility` class. The agent folder is included to allow state manipulation and agent forwarding. The `AgentCodeUtility` class provides public utility methods that are available to agents on each host. An agent can retrieve the name of the local host using the `getLocation` method. `fwdAgent` sends a serialized `agentPacket` to another host when requested by the agent. `getState` and `saveState` retrieve and store the state field of the agent folder. The AgentState class contains three fields:

```
public class AgentState implements Serializable {
   public boolean initialVisit;
   public String  tempDestination;
   public String  collectedData;
}
```

`initialVisit` is set to true when first executing on the originator's host. This allows the agent to know when it has returned from its travels. `tempDestination` stores a variable host name permitting the agent to alter its itinerary based on collected destinations. Information collected along the way is stored in the `collectedData` field. The `getDestination` and `getData` methods read from a file to simulate accessing a database.

B. JAVA IMPLEMENTATION OF SECURE MOBILE AGENT TRANSMISSION PROTOCOL

The certificate authority is simulated by the `BuildCertificate` class included on each host. `BuildCertificate` creates a certificate and inserts the local host's name and DSA public key. DSA digital signature support is provided by the Java Cryptography Architecture. This is used to simulate the model's use of a public key cryptography standard, such as RSA, which is not implemented by the Java 1.1 API. (It is available in JSafe 1.0, though.)

`BuildCertificate` then encodes the certificate by signing it with the certificate authority's DSA private key. All hosts are manually preloaded with the certificate authority's DSA public key simulating CA access.

Following a `fwdAgent` request and prior to sending an agent folder, the destination must be authenticated. This is accomplished in the private `destinationAuthenticates` method of the `fwdAgent` utility method. See Figure 5. The local signed certificate is retrieved from the `cert` file, loaded in an agent packet and sent to port 6011 on the destination host. At the destination host, the `AgentPacketServer` reads the incoming packet and decodes the signed certificate.

It then checks to see if the host name in the certificate is a member of a set of trusted hosts. If it is, a reply in the form of an agent packet including the destination host's signed certificate is sent to port 6012 of the source host. The `destinationAuthenticates` method continues at the souce host by reading the incoming reply and decoding the signed certificate. A check is made to ensure the received certificate's host name is the same as the request destination. If it is, the destination has authenticated and the agent folder can be forwarded.

The agent folder is signed with the local host's DSA private key and sent to port 6013 of the destination host. Here the `AgentPacketServer` continues on the destination host and decodes the agent packet by verifying the signature.

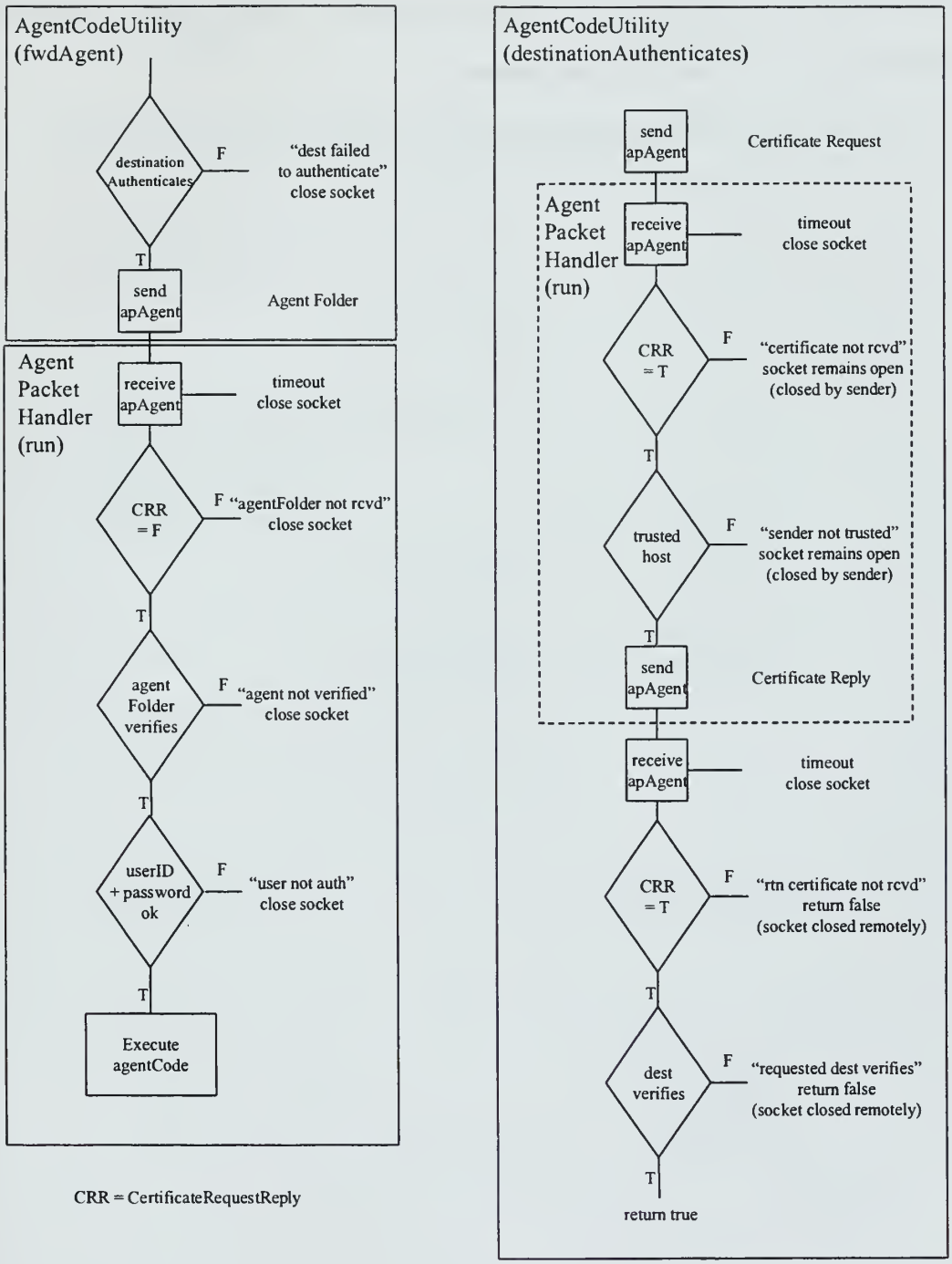

Figure 5. `fwdAgent` Authentication Protocol

A user identification check is then made. This simulates the possible need for an agent originator to maintain account information at the destination host. Following this check the agent is executed as described above.

The classes and interfaces for this implementation are shown in Figure 6.

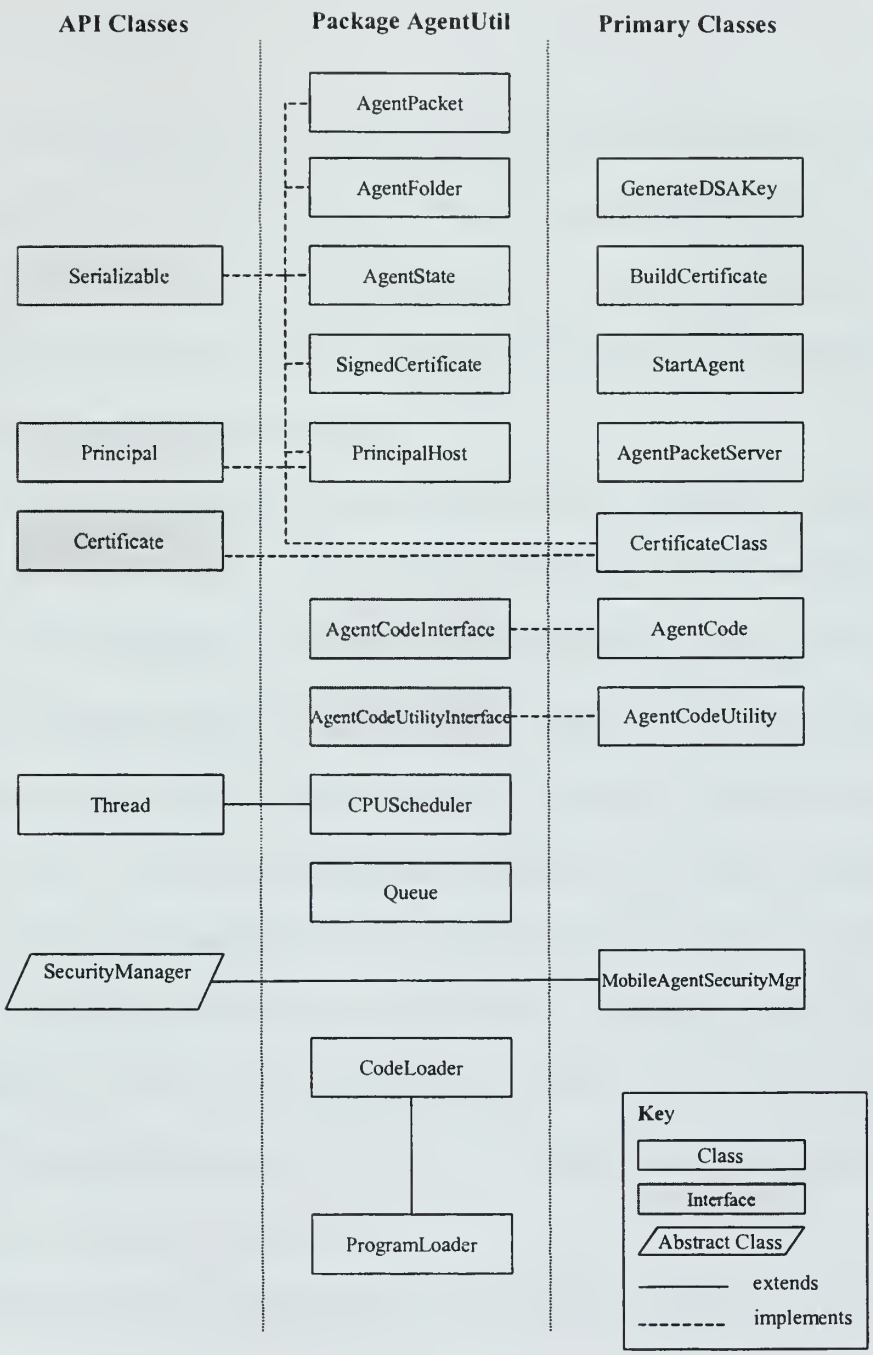

Figure 6. Classes and Interfaces

V. A MILITARY SCENARIO FOR MOBILE AGENTS

A. DATA MINING

The following is a military scenario exemplifying, agent activation, dispatching and returning home. It is a data mining scenario, using agents to search a network of intelligence-related databases. The data being mined is specific to intelligence, but can be generalized to any information type. Other types of military related information types include weather, cartography and logistics.

The players in this scenario are a Joint Task Force Headquarters, a Theater Intelligence Headquarters and two Sub-Regional Intelligence Centers. See Figure 7. The Joint Task Force Headquarters is an integrated service composition charged with tactical command and control of a specific military mission. Examples of military missions include strike operations, non-combatant evacuation operations, amphibious assaults, disaster relief, etc. The Theater Intelligence Headquarters is the regional intelligence information manager. It provides operational tasking for many Sub-Regional Intelligence Centers. The Sub-Regional Intelligence Centers are basic intelligence data collection agencies located on land, air and sea based platforms. They contain various sensor types and provide specialized intelligence information. This information can be based on radar data, imagery, visual contacts, etc.

The Joint Task Force Headquarters in this scenario is planning an air strike operation targeted at some enemy location. During this strike mission, a number of friendly aircraft will fly over the area of interest and drop a payload of bombs intent on destroying the target. Of concern to the Joint Task Force Headquarters is an enemy

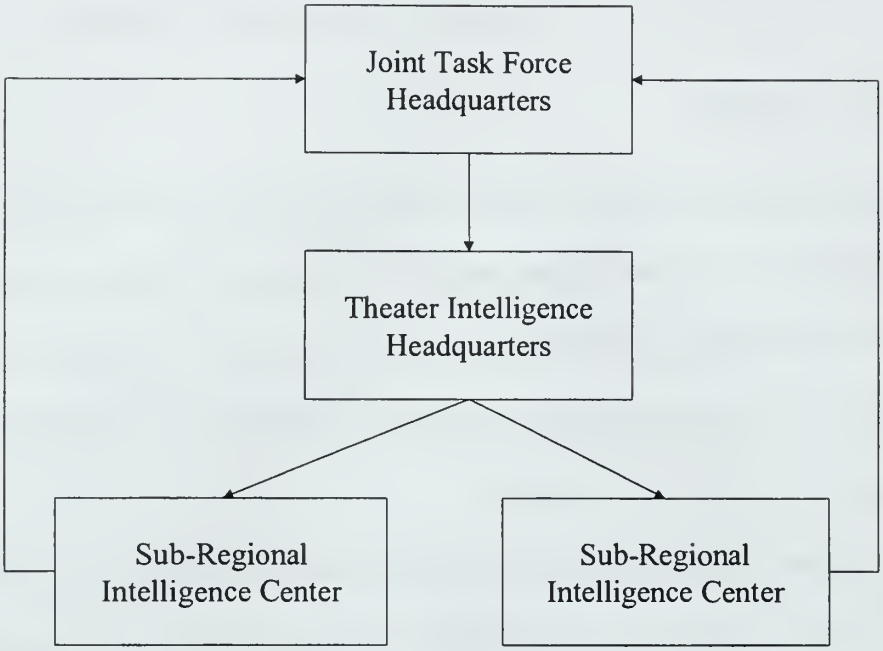

Figure 7. Military Data Mining Scenario

airfield near the target which could launch a defensive sortie of say, F-4 fighter aircraft. This enemy sortie could interfere with the mission.

Prior to ordering the friendly aircraft to launch, the Joint Task Force Headquarters needs to be aware of any enemy aircraft activity at the airfield in question.

A request is made by a user at the Joint Task Force Headquarters to find out if any F-4 fighter aircraft activity has been observed at the enemy airfield. An agent is activated and given the following tasks:

- Go to the Theater Intelligence Headquarters and find out if enemy F-4 fighter aircraft can be launched from the airfield.
- Find out where this activity can first be observed and go there.
- Collect any recent F-4 related activity and return with the collected details.

The agent dispatches to the Theater Intelligence Headquarters and receives a positive confirmation to the first query. The second query is responded to by advising the

agent to visit two Sub-Regional Intelligence Centers tasked with covering the airfield. These include a land based radar station and an overhead sensor center. The agent then dispatches itself to each suggested location and queries the local databases for indication of any F-4 aircraft activity. One of the agents soon sees an F-4 launch report associated with the airfield and immediately returns home to report the activity along with launch time and number of aircraft launched. The other agent also retrieves a similar launch report and returns home confirming the earlier report.

By using the trusted mobile agent model described in Chapter III, the Joint Task Force Headquarters can, with confidence, make critical decisions based on the data collected at the trusted remote sensor sites. They are assured the results to their queries are authentic and accurate. They also know the data has been kept private and not disclosed to the enemy based on the level of public key encryption used in the implementation.

VI. CONCLUSION

This thesis has demonstrated that a trusted mobile agent model can be useful in realizing agent applications in the military. This is based upon the assumption that all useful agents originate from a trusted host and are only forwarded to other trusted hosts. This is a valid assumption in the military world where it is common to communicate with a predetermined set of agencies among which some degree of trust exists. This model requires that each host maintain a list of hosts that it can expect will send only trustworthy agents and not malicious code.

The approach employs the Secure Mobile Agent Transmission Protocol using public key cryptography. One version of it requires storing host public keys on each host. Another version allows more flexible re-keying, but depends on certificates signed by a certification authority. This version is useful when a host's public key must change, for example, if its private key has been compromised or revoked. Since hosts do not store each other's public keys, re-keying a host merely amounts to issuing a new certificate for it. Of course re-keying the certificate authority would affect all hosts.

Although a mobile agent system could be implemented in any programming language, Java is a natural choice with its built-in support for networking and dynamic class loading. Java exhibits platform independence. This is crucial for these software agents that are mobile and execute in a heterogeneous domain, like the Internet. Java also provides serialization for easy storage of object state, allowing data to be transported from host to host.

Another agent-based military scenario can be found in (Edmiston, 1998). Their scenario is similar and is used to describe a framework for an agent-based decision making application.

This thesis attempts to show that a client can dispatch an autonomous agent into an open network of known trusted databases and upon return of the agent, have confidence in its results. Obviously this is important in military operational environments where results returned by an agent may be used in making decisions that could endanger human safety or affect delicate foreign relations.

APPENDIX A. MOBILE AGENT CODE

1. INTRODUCTION

The mobile agent implementation is based on my advisor's active network design and much of the source code is rooted in the exercises and projects from the CS3973 Advanced Object-Oriented Programming in Java course at the Naval Postgraduate School, Monterey, California.

The implementation is a certificate-based authenticate-forward-authenticate protocol. The code differs slightly from the Secure Mobile Agent Transmission Protocol model in that Host A's certificate is forwarded in the initial request verses being forwarded in the final agent packet which contains the agent folder.

The mobile agent code is grouped into two sections, the primary code and the AgentUtil package.

The primary code section contains the classes that generate DSA keys and build the certificates used in the model implementation. Also included are the Agent Packet Server, Start Agent, Agent Code, Agent Code Utility and Mobile Agent Security Manager classes.

The AgentUtil section contains the classes and interfaces used in support of the primary code.

The third section of this appendix contains the data and destination files used in simulating databases in the network. Also included is a printout of the originating host's display following a successful run of the agent.

2. PRIMARY CODE

```
//**********************************************************************
// File:          AgentCode.java
// Name:          LT Roy J. Virden
// Date:          25 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//**********************************************************************

import java.io.*;
import agentUtil.*;


//---------------------------------------------------------------------
//   AgentCode class
//---------------------------------------------------------------------
public class AgentCode implements AgentCodeInterface {


   //--------------------
   // exec
   //--------------------
   public void exec(AgentFolder af, AgentCodeUtilityInterface u)
```

```
                                                throws Exception {

AgentState as = u.getState(af);

if ("ro140203.cc.nps.navy.mil".equals(u.getLocation())) {
   if (as.initialVisit == true) {   // initial visit = true
      as.initialVisit = false;
      System.out.println("agentCode: I'm at Home (" +
                         u.getLocation() + ")");
      af.agentStateBA = u.saveState(as);
      System.out.println("agentCode: forwarding agent to TIHQ");
      u.fwdAgent(af, "ro140204.cc.nps.navy.mil");
   }
   else {
      System.out.println("agentCode: I have returned Home (" +
                         u.getLocation() + ")");
      System.out.println("agentCode: returned data requested:");
      System.out.println("agentCode: " + as.collectedData);
   }
}

// we know location of TIHQ
if ("ro140204.cc.nps.navy.mil".equals(u.getLocation())) {
   System.out.println("agentCode:  I'm at TIHQ (" +
                      u.getLocation() + ")");

   // here we run through a database and an agent is forwarded
   // to each site able to observe F4 activity.
   // agentState.destination field is used


   as.tempDestination = u.getDestination("dest/destination1");
   af.agentStateBA    = u.saveState(as);
   System.out.println("agentCode:  forwarding agent to SRIC (" +
                      as.tempDestination + ")");
   u.fwdAgent(af, as.tempDestination);

   as.tempDestination = u.getDestination("dest/destination2");
   af.agentStateBA    = u.saveState(as);
   System.out.println("agentCode:  forwarding agent to SRIC (" +
                      as.tempDestination + ")");
   u.fwdAgent(af, as.tempDestination);
}

// compare current location against agentState.destination field
if ((as.tempDestination).equals(u.getLocation())) {
   System.out.println("agentCode:  I'm at SRIC (" +
                      u.getLocation() + ")");

   // search a database and store data field in agentState.data
   // for each data entry related to F4 activity
   // and then forward the agent home

   as.collectedData = u.getData();
   af.agentStateBA  = u.saveState(as);
   System.out.println("agentCode:  forwarding agent to Home");
   u.fwdAgent(af, "ro140203.cc.nps.navy.mil");       // go home
```

```
        }
    }
}
```

```
//*********************************************************************
// File:         AgentCodeUtility.java
// Name:         LT Roy J. Virden
// Date:         29 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import java.net.*;
import java.io.*;
import java.util.*;
import java.security.*;
import agentUtil.*;


//---------------------------------------------------------------------
//  AgentCodeUtility class
//---------------------------------------------------------------------
public class AgentCodeUtility implements AgentCodeUtilityInterface {


   //--------------------
   // getLocation returns the name of localHost
   //--------------------
   public String getLocation() {
      String localHost = null;
      try {
         localHost = InetAddress.getLocalHost().getHostName();
      } catch (Exception e) {
         System.out.println(e);
      }
      return localHost;
   }


   //--------------------
   // fwdAgent sends a serialized agent packet to the requested host
   //--------------------
   public void fwdAgent(AgentFolder af, String destination)
                                          throws Exception {

       try {

         //----------
         // create AgentPacket and authenticate destination
         //----------
         AgentPacket ap = new AgentPacket(true);
         if (destinationAuthenticates(destination, ap) == true) {

            //----------
            // set AgentPacket to agent, serialize, sign and send
            //----------
            ap.certificateRequestReply = false;  // flag agent packet
            ap.signedCertificateBA = null;       // remove certificate

            byte[] afBA = serialize(af);
```

```
            //----------
            // get privateKey, sign agentFolder byte[], pack & send it
            //----------
            FileInputStream fisPrivateA = new FileInputStream("keys/" +
                                          this.getLocation() +
                                          "_privateKey");
            ObjectInputStream oisPrivateA =
               new ObjectInputStream(fisPrivateA);
            Signature dsa = Signature.getInstance("DSA");
            dsa.initSign((PrivateKey)oisPrivateA.readObject());
            dsa.update(afBA);
            byte [] afBASignature     = dsa.sign();
            ap.agentFolderBA          = afBA;
            ap.agentFolderBASignature = afBASignature;
            byte[] apBA               = serialize(ap);
            sendAPBA(destination, 6013, apBA);

            System.out.println("Destination Authenticated");
            System.out.println("agentPacket forwarded to " +
                               destination);

         } // end if (destinationAuthenticates(destination,ap) == true)
         else System.out.println(destination +
                                 " failed to authenticate!");

      } catch (Exception e) {
         System.out.println(e);

      }
   }


   //--------------------
   // desitinationAuthenticates method
   //--------------------
   private static boolean destinationAuthenticates(String dest,
                                                   AgentPacket ap)
                                            throws Exception {
      try {

         //----------
         // get and send certificate to destination
         //----------
         FileInputStream   fisSCBA = new
            FileInputStream("certs/SCBA_Certificate");
         ObjectInputStream oisSCBA = new ObjectInputStream(fisSCBA);
         ap.signedCertificateBA    = ((byte[])oisSCBA.readObject());
         byte[]            apBA    = serialize(ap);
         sendAPBA(dest, 6011, apBA);

// (goto AgentPacketServer.java)

         //----------
         // read returning AgentPacket reply, deserialize and cast
         //----------
         byte[] incomingObject = new byte[65507];
         ServerSocket ss2 = new ServerSocket(6012);
```

```
      Socket s2 = ss2.accept();
      receiveAPBA(incomingObject, s2);
      ss2.close();
      ap = (AgentPacket)(deserialize(incomingObject));

      //----------
      // handle certificate
      //----------
      if (ap.certificateRequestReply == true) {
         CertificateClass c = new CertificateClass();
         ByteArrayInputStream bais =
            new ByteArrayInputStream(ap.signedCertificateBA);
         c.decode(bais);

         //----------
         // check that socket host requested verifies
         //----------
         if (c.principal.equals(dest)) {

            //----------
            // save received public key to file
            // (may be used in RSA implementation)
            //----------
            FileOutputStream   fosPublic =
               new FileOutputStream("keys/" +
                                    c.principal.getName() +
                                    "_publicKey");
            ObjectOutputStream oosPublic = new
               ObjectOutputStream(fosPublic);
            oosPublic.writeObject(c.publicKey);
            fosPublic.close();
            oosPublic.close();

            return true;  // return from successful authentication

         } // end if (c.serverName.equals(dest)
         else {
            System.out.println("ACU: Cert principal does not match
                               destination!");
            return false;
         }

      } // end if (ap.certificateRequestReply == true)
      else {
         System.out.println("Cert reply not received!");
         return false;
      }

   } catch (Exception e) {
      System.out.println(e);
      return false;
   }

} // end method


//----------
```

```
// sendAPBA method
//----------
private static void sendAPBA(String destination,
                             int port,
                             byte[] apBA)
                                                 throws Exception {
   Socket s = new Socket(destination, port);
   OutputStream os = s.getOutputStream();
   os.write(apBA);
   os.close();
   s.close();
}


//----------
// receiveAPBA method
//----------
private static byte[] receiveAPBA(byte[] incomingObject, Socket s)
                                                 throws Exception {
   InputStream is = s.getInputStream();
   int cc;
   int len = 0;
   while ((cc = is.read()) != -1)
      incomingObject[len++] = (byte)cc;
   is.close();
   s.close();
   return incomingObject;
}


//--------------------
// serialize method  (from Object to byte[])
//--------------------
private static byte[] serialize(Object obj) throws Exception {
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   ObjectOutputStream    oos  = new ObjectOutputStream(baos);
   oos.writeObject(obj);
   oos.flush();
   oos.close();
   return baos.toByteArray();
}


//--------------------
// deserialize method (from byte[] to Object)
//--------------------
private static Object deserialize(byte[] ba) throws Exception {
   ByteArrayInputStream bais = new ByteArrayInputStream(ba);
   ObjectInputStream    ois  = new ObjectInputStream(bais);
   ois.close();
   return ois.readObject();
}


//--------------------
// getState method  (from AgentFolder.agentStateBA to AgentState)
//--------------------
```

```
public AgentState getState(AgentFolder af) throws Exception {
   return (AgentState)(deserialize(af.agentStateBA));
}


//--------------------
// saveState method (from AgentState to byte[])
//--------------------
public byte[] saveState(AgentState as) throws Exception {
   return serialize(as);
}


//--------------------
// getDestination method
//--------------------
public String getDestination(String nextDest) throws Exception {
      FileInputStream fis = new FileInputStream(nextDest);
      DataInputStream dis = new DataInputStream(fis);
      return dis.readLine();
}


//--------------------
// getData method
//--------------------
public String getData() throws Exception {
      FileInputStream fis = new FileInputStream("data/IntelData");
      DataInputStream dis = new DataInputStream(fis);
      return dis.readLine();
}
```

```
//*********************************************************************
// File:         AgentPacketServer.java
// Name:         LT Roy J. Virden
// Date:         29 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5 JDK1.1.2
//*********************************************************************

import java.io.*;
import java.net.*;
import agentUtil.*;
import java.security.*;


//---------------------------------------------------------------------
//  AgentPacketServer class      Command: java AgentPacketServer &
//---------------------------------------------------------------------
public class AgentPacketServer {


   //--------------------
   // main sets the system security manager.  Then it listens
   //    for a connection request, on port 6011, for an agent
   //    folder.  A new Thread is created and
   //    added to the CPU scheduler which handles the incoming
   //    agent packets.
   //--------------------
   public static void main(String args[]) throws Exception {

      SecurityManager secMgr = new MobileAgentSecurityMgr();
      System.setSecurityManager(secMgr);

      try {

         CPUScheduler cpuScheduler = new CPUScheduler(50);
         cpuScheduler.start();

         Thread MobileAgentThread;

         // allows multiple threaded clients on local port 6011

         ServerSocket ss1 = new ServerSocket(6011);

         while (true) {
            // accept connection from client
            MobileAgentThread = new AgentPacketHandler(ss1.accept());
            cpuScheduler.addThread(MobileAgentThread);
            MobileAgentThread.start();
         }

      } catch (IOException e) { System.out.println(e);}
   }
}
```

```
//---------------------------------------------------------------------
//  AgentPacketHandler class
//---------------------------------------------------------------------
class AgentPacketHandler extends Thread {

   Socket sock;


   //--------------------
   // constructor
   //--------------------
   public AgentPacketHandler (Socket s) {
      this.sock = s;
   }


   //--------------------
   // threaded run method
   //--------------------
   public void run() {

      try {

         //----------
         // read incoming AgentPacket request, deserialize and cast
         //----------
         byte[] incomingObject = new byte[10000];
         receiveAPBA(incomingObject, sock);
         AgentPacket ap = (AgentPacket)(deserialize(incomingObject));

System.out.println("APS: ap.certificateRequestReply = " +
ap.certificateRequestReply);

         //----------
         // handle certificate
         //----------
         if (ap.certificateRequestReply == true) {
            CertificateClass c = new CertificateClass();
            ByteArrayInputStream bais =
               new ByteArrayInputStream(ap.signedCertificateBA);
            c.decode(bais);

            //----------
            // check list of trusted hosts
            //----------
            if ( c.principal.equals( "ro140203.cc.nps.navy.mil" ) ||
                 c.principal.equals( "ro140204.cc.nps.navy.mil" ) ||
                 c.principal.equals( "ro140206.cc.nps.navy.mil" ) ||
                 c.principal.equals( "ro140207.cc.nps.navy.mil" )   ) {

               //----------
               // set AgentPacket reply, get cert, serialize & send
               //----------
               ap.certificateRequestReply = true;
               FileInputStream   fisSCBA  = new
                  FileInputStream("certs/SCBA_Certificate");
               ObjectInputStream oisSCBA  = new
```

```
                ObjectInputStream(fisSCBA);
            ap.signedCertificateBA     =
                ((byte[])oisSCBA.readObject());
            byte[] apBA                = serialize(ap);
            sendAPBA(c.principal.toString(), 6012, apBA);

// (goto AgentCodeUtility.java)

            //----------
            // read incoming AgentPacket byte[], deserialize & cast
            //----------
            ServerSocket ss3 = new ServerSocket(6013);
            Socket       s3  = ss3.accept();
            receiveAPBA(incomingObject, s3);
            ss3.close();
            ap = (AgentPacket)(deserialize(incomingObject));

            //----------
            // handle agentFolder
            //----------
            if (ap.certificateRequestReply == false) {

               if (verifyAgentFolder(ap, c) == true) {
                  System.out.println("APS: agentFolder verified!");

                  AgentFolder af =
                     (AgentFolder)(deserialize(ap.agentFolderBA));

                  //----------
                  // check list of trusted originating hosts
                  // [optional]
                  //----------
                  if (af.userID.equals("rjvirden")) {

                     //----------
                     // create agent utilities and get instance of
                     //   agentFolder code
                     //----------
                     AgentCodeUtilityInterface util = new
                        AgentCodeUtility();
                     CodeLoader codeLoader = new CodeLoader();
                     AgentCodeInterface agentCode =
                        codeLoader.getActive(af.agentCodeBA);
                     agentCode.exec(af, util);   // start agent code

                  } // end if (af.userID.equals("rjvirden"))
                  else System.out.println("APS: User not
                                          authenticated!");

               } // end if (verifyAgentFolder(ap) == true)
               else System.out.println("APS: Agent does not
                                       verify!");

            } // end if (ap.certificateRequestReply == false)
            else System.out.println("APS: Agent Folder not
                                    received!");
```

```
         } // end if(c.principal.equals("ro140204.cc.nps.navy.mil"))
         else System.out.println("APS: " + c.principal.getName() +
                                 " not trusted!");

      } // end if (ap.certificateRequestReply == true)
      else System.out.println("APS: Certificate Request not
                              received!");

   } catch (Exception e) {
      System.out.println("APS run method exception: " + e);
   }

} // end run


//----------
// sendAPBA method
//----------
private static void sendAPBA(String destination,
                             int port,
                             byte[] apBA)
                                                 throws Exception {
   Socket       s  = new Socket(destination, port);
   OutputStream os = s.getOutputStream();
   os.write(apBA);
   os.close();
   s.close();
}


//----------
// receiveAPBA method
//----------
private static byte[] receiveAPBA(byte[] incomingObject, Socket s)
                                                   throws Exception {
   InputStream is = s.getInputStream();
   int cc;
   int len = 0;
   while ((cc = is.read()) != -1)
      incomingObject[len++] = (byte)cc;
   is.close();
   s.close();
   return incomingObject;
}


//--------------------
// deserialize method
//--------------------
private static Object deserialize(byte[] ba) throws Exception {
   ByteArrayInputStream bais = new ByteArrayInputStream(ba);
   ObjectInputStream    ois  = new ObjectInputStream(bais);
   ois.close();
   return ois.readObject();
}
```

```
    //--------------------
    // serialize method
    //--------------------
    private static byte[] serialize(Object obj) throws Exception {
       ByteArrayOutputStream baos = new ByteArrayOutputStream();
       ObjectOutputStream    oos  = new ObjectOutputStream(baos);
       oos.writeObject(obj);
       oos.flush();
       oos.close();
       return baos.toByteArray();
    }


    //--------------------
    // verifyAgentFolder method  get sender's publicKey and verify
    //--------------------
    private static boolean verifyAgentFolder(AgentPacket ap,
                                             CertificateClass c)
                                                 throws Exception {
       Signature dsa    = Signature.getInstance("DSA");
       dsa.initVerify(c.publicKey);
       dsa.update(ap.agentFolderBA);
       boolean verified = dsa.verify(ap.agentFolderBASignature);
       return verified;
    }

} // end class
```

```
//*********************************************************************
// File:         BuildCertificate.java
// Name:         LT Roy J. Virden
// Date:         24 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import agentUtil.*;
import java.io.*;
import java.security.*;
import java.net.*;


//---------------------------------------------------------------------
//  BuildCertificate class
//---------------------------------------------------------------------
class BuildCertificate {


   //--------------------
   // main
   //--------------------
   public static void main(String [] args) throws Exception {

      //----------
      // get local host name and initialize principals
      //----------
      AgentCodeUtilityInterface util = new AgentCodeUtility();
      PrincipalHost ph = new PrincipalHost(util.getLocation());
      PrincipalHost pCA = new PrincipalHost("CA");

      //----------
      // create certificate and get members
      //----------
      CertificateClass c          = new CertificateClass();
      c.principal                 = ph;
      FileInputStream fisPublic   = new FileInputStream("keys/" +
                                          ph.getName() + "_publicKey");
      ObjectInputStream oisPublic = new ObjectInputStream(fisPublic);
      c.publicKey                = ((PublicKey)oisPublic.readObject());
      c.guarantor                = pCA;
      c.format                   = "DSA";
      fisPublic.close();
      oisPublic.close();

      //----------
      // get outputStream to file and encode certficate.
      //   this implementation of encode performs a DSA signing. the
      //   resulting byte array is then written to the output stream
      //   specified as an input parameter (a file in this case).
      //----------
      FileOutputStream fosSCBA   = new
         FileOutputStream("certs/SCBA_Certificate");
      c.encode(fosSCBA);
   }
}
```

```
//*********************************************************************
// File:         CertificateClass.java
// Name:         LT Roy J. Virden
// Date:         25 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import agentUtil.*;
import java.io.*;
import java.security.*;


//---------------------------------------------------------------------
//  CertificateClass class  (must be able to access keys and certs
//                           directories)
//---------------------------------------------------------------------
public class CertificateClass implements Certificate, Serializable {

   public Principal principal;
   public PublicKey publicKey;
   public Principal guarantor;
   public String    format;


   //--------------------
   // constructor
   //--------------------
   public CertificateClass() {
      this.super();
   }


   //--------------------
   // getFormat method returns encoding format
   //--------------------
   public String getFormat() {
      return this.format;
   }


   //--------------------
   // getGuarantor method returns guarantor
   //--------------------
   public Principal getGuarantor() {
      return this.guarantor;
   }


   //--------------------
   // getPrincipal method returns principal
   //--------------------
   public Principal getPrincipal() {
      return this.principal;
   }
```

```
//--------------------
// getPublicKey method returns publicKey
//--------------------
public PublicKey getPublicKey() {
   return this.publicKey;
}


//--------------------
// toString method returns a String
//--------------------
public String toString(boolean bool) {
   return this.getPrincipal().getName();
}


//--------------------
// encode method encodes a certificate to an outputStream
//--------------------
public void encode(OutputStream fosSCBA) {

   try {

      //----------
      // serialize 'this' certificate to a byte array
      //----------
      byte[] cBA  = serialize(this);

      //----------
      // get CA privateKey, read byte[] message and sign it
      //----------
      FileInputStream   fisPrivateCA =
         new FileInputStream("keys/" + this.guarantor.getName() +
                             "_privateKey");
      ObjectInputStream oisPrivateCA = new
         ObjectInputStream(fisPrivateCA);
      Signature dsa = Signature.getInstance("DSA");
      dsa.initSign((PrivateKey)oisPrivateCA.readObject());
      dsa.update(cBA);
      byte [] cBASignature = dsa.sign();
      fisPrivateCA.close();
      oisPrivateCA.close();

      //----------
      // create signed certificate, get members and serialize
      //----------
      SignedCertificate sc      = new SignedCertificate();
      sc.certificateBA          = cBA;
      sc.certificateBASignature = cBASignature;
      byte[] scBA               = serialize(sc);

      //----------
      // write certificate to file
      //----------
      ObjectOutputStream oosSCBA = new ObjectOutputStream(fosSCBA);
      oosSCBA.writeObject(scBA);
```

```
      fosSCBA.close();
      oosSCBA.close();

   } catch (Exception e) {
      System.out.println("CertificateClass.encode exception: " + e);
   }
}


//--------------------
// decode method decodes a certificate from an inputStream
//--------------------
public void decode(InputStream bais) {
   try {

      //----------
      // deserialize inputStream (ap.signedCertificateBA in this
      //    case)
      //----------
      ObjectInputStream ois = new ObjectInputStream(bais);
      SignedCertificate sc  = (SignedCertificate)(ois.readObject());
      boolean verified      = verifyCertificate(sc);
      ois.close();

      //----------
      // if certificate verifies, deserialize and load 'this'
      //   certificate
      //   with incoming certificate fields
      //----------
      if (verified) {
         CertificateClass tempCert =
            (CertificateClass)(deserialize(sc.certificateBA));

         this.principal = tempCert.principal;
         this.publicKey = tempCert.publicKey;
         this.guarantor = tempCert.guarantor;
         this.format    = tempCert.format;
      }
   } catch (Exception e) {
      System.out.println("CertificateClass.decode exception: " + e);
   }
}


//--------------------
// serialize method  (from Object to byte[])
//--------------------
private static byte[] serialize(Object obj) throws Exception {
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   ObjectOutputStream    oos  = new ObjectOutputStream(baos);
   oos.writeObject(obj);
   oos.flush();
   oos.close();
   return baos.toByteArray();
}
```

```
    //--------------------
    // deserialize method  (from byte[] to Object)
    //--------------------
    private static Object deserialize(byte[] ba) throws Exception {
       ByteArrayInputStream bais = new ByteArrayInputStream(ba);
       ObjectInputStream    ois  = new ObjectInputStream(bais);
       ois.close();
       return ois.readObject();
    }


    //--------------------
    // verifyCertificate method (get CA publicKey and verify)
    //--------------------
    private static boolean verifyCertificate(SignedCertificate sc)
                                                    throws Exception {
       FileInputStream   fisPublicCA =
          new FileInputStream("keys/" + "CA" + "_publicKey");
       ObjectInputStream oisPublicCA = new
          ObjectInputStream(fisPublicCA);
       Signature dsa = Signature.getInstance("DSA");
       dsa.initVerify((PublicKey)oisPublicCA.readObject());
       dsa.update(sc.certificateBA);
       boolean verified = dsa.verify(sc.certificateBASignature);
       return verified;
    }
}
```

```
//*********************************************************************
// File:         GenerateDSAKeys.java
// Name:         LT Roy J. Virden
// Date:         29 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import java.io.*;
import java.security.*;
import java.net.*;
import agentUtil.*;


//---------------------------------------------------------------------
//  GenerateDSAKeys class
//---------------------------------------------------------------------
class GenerateDSAKeys {


   //----------
   // main
   //----------
   public static void main(String [] args) {

      //----------
      // generate and store public and private keys
      //   Only produce one CA keypair and copy to other machines
      //----------
      AgentCodeUtilityInterface util = new AgentCodeUtility();
      generateAndStoreKeys(util.getLocation(),  util.getLocation() +
                           "seed string");
      generateAndStoreKeys("CA", "CA seed string");
   }


   //----------
   // generateAndStoreKeys
   //----------
   public static void generateAndStoreKeys(String name,
                                           String inputSeedString) {

      try {

         //----------
         // create files
         //----------
         FileOutputStream fosPrivate   = new FileOutputStream("keys/" +
                                             name + "_privateKey");
         FileOutputStream fosPublic    = new FileOutputStream("keys/" +
                                             name + "_publicKey");
         ObjectOutputStream oosPrivate = new
            ObjectOutputStream(fosPrivate);
         ObjectOutputStream oosPublic  = new
            ObjectOutputStream(fosPublic);
```

```
            //----------
            // random number generator seed
            //----------
            String seedString = inputSeedString;
            byte[] seed       = seedString.getBytes();

            //----------
            // generate keys
            //----------
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(1024, new SecureRandom(seed));
            KeyPair pair = keyGen.generateKeyPair();

            //----------
            // write keys to file
            //----------
            oosPrivate.writeObject(pair.getPrivate());
            oosPublic.writeObject(pair.getPublic());
            oosPrivate.close();
            oosPublic.close();

        } catch(NoSuchAlgorithmException e) {
            System.out.println("NoSuchAlgorithmException");
        } catch(java.io.IOException e) {
            System.out.println("IOException");
        }
    }
}
```

```
//*********************************************************************
// File:         MobileAgentSecurityMgr.java
// Name:         LT Roy J. Virden
// Credit:       This class is an adaptation of Professor Volpano's
//               code from the CS3973 Advanced Object-Oriented
//               Programming in Java course.
// Date:         29 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import java.io.*;


//---------------------------------------------------------------------
//  MobileAgentSecurityMgr class
//---------------------------------------------------------------------
class MobileAgentSecurityMgr extends SecurityManager {

        protected MobileAgentSecurityMgr() {
                super();
        }

        // allow AgentCodeUtilityInterface fwdBytes to open a socket

        public void checkConnect(String host, int port) { }
        public void checkConnect(String host, int port, Object o) { }

        // allow creation of a new ClassLoader object

        public void checkCreateClassLoader() { }

        // allow active node server to listen on a port

        public void checkAccept(String host, int port) { }

        // prevent ExceptionInInitializerError at server startup

        public void checkAccess(Thread t) { }
        public void checkAccess(ThreadGroup g) { }
        public void checkListen(int port) { }
        public void checkLink(String lib) { }
        public void checkPropertyAccess(String k) { }

        // allow getInputStream().read(buf) to receive incoming class

        public void checkRead(FileDescriptor fd) { }

        // allow getOutputStream() to forward a class

        public void checkWrite(FileDescriptor fd) { }

        public void checkRead(String file) {
           if (! ( file.endsWith("java.security")    ||
                   file.endsWith("_publicKey")       ||
                   file.endsWith("_privateKey")      ||
                   file.endsWith("SCBA_Certificate") ||
```

```
                file.endsWith("destination1")     ||
                file.endsWith("destination2")     ||
                file.endsWith("IntelData")                ) )
            throw new SecurityException(file + " unreadable");
        }

        public void checkWrite(String file) {
            if (! ( file.startsWith("/tmp/")    ||
                file.endsWith("_publicKey")    ) )
            throw new SecurityException(file + " unwritable");
        }
}
```

```
//*********************************************************************
// File:         StartAgent.java
// Name:         LT Roy J. Virden
// Date:         29 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

import java.io.*;
import java.net.*;
import java.util.*;
import agentUtil.*;
import java.security.*;


//-------------------------------------------------------------------
//  StartAgent class injects AgentCode.class into the home agent
//     server.       Command: java StartAgent
//-------------------------------------------------------------------
public class StartAgent {


   //--------------------
   // main
   //--------------------
   public static void main( String argv[] ) throws Exception {

      //----------
      // access AgentCodeUtility methods
      //----------
      AgentCodeUtilityInterface util = new AgentCodeUtility();

      //----------
      // build agentfolder
      //----------
      AgentFolder af      = new AgentFolder();
      af.userID           = "rjvirden";
      af.password         = "password";

      // get bytecode
      FileInputStream fis = new FileInputStream("AgentCode.class");
      byte[] bytecode     = new byte[fis.available()];
      fis.read(bytecode);
      System.out.println("uploading agentCode:");
      af.agentCodeBA      = bytecode;

      AgentState as       = new AgentState();
      as.initialVisit     = true;
      af.agentStateBA     = util.saveState(as);

      af.encrypted        = false;   // true indicates agentData
                                     //   encrypted with sessionKey
      af.sessionKeyBA     = null;    // used to encrypt agentData
                                     //   member
```

```
      //----------
      // get local host name for injection
      //----------
      String destination = util.getLocation();    // initial home host
      util.fwdAgent(af, destination);
      System.out.println("StartAgent finished");

// (goto AgentCodeUtility.java)

   }
}
```

3. AGENTUTIL PACKAGE

```
//***************************************************************************
// File:          AgentCodeInterface.java
// Name:          LT Roy J. Virden
// Date:          10 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//***************************************************************************

package agentUtil;


//---------------------------------------------------------------------
//  AgentCodeInterface interface
//---------------------------------------------------------------------
public interface AgentCodeInterface {


   //--------------------
   // exec
   //--------------------
   public void exec(agentUtil.AgentFolder b,
                    agentUtil.AgentCodeUtilityInterface u)
                                                 throws Exception;

}
```

```
//*********************************************************************
// File:         AgentCodeUtilityInterface.java
// Name:         LT Roy J. Virden
// Date:         17 December 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;


//---------------------------------------------------------------------
//  AgentCodeUtilityInterface interface
//---------------------------------------------------------------------
public interface AgentCodeUtilityInterface {


   //--------------------
   // getLocation
   //--------------------
   public String getLocation();


   //--------------------
   // fwdAgent
   //--------------------
   public void fwdAgent(AgentFolder b, String dest) throws Exception;


   //--------------------
   // getState
   //--------------------
   public AgentState getState(AgentFolder af) throws Exception;


   //--------------------
   // saveState
   //--------------------
   public byte[] saveState(AgentState as) throws Exception;


   //--------------------
   // getDestination method
   //--------------------
   public String getDestination(String nextDest) throws Exception;


   //--------------------
   // getData method
   //--------------------
   public String getData() throws Exception;


}
```

```
//*********************************************************************
// File:          AgentFolder.java
// Name:          LT Roy J. Virden
// Date:          16 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;
import java.io.*;


//---------------------------------------------------------------------
//  AgentFolder class
//---------------------------------------------------------------------
public class AgentFolder implements Serializable {

   public String  userID;
   public String  password;
   public byte[]  agentCodeBA;
   public byte[]  agentStateBA;
   public boolean encrypted = false;
   public byte[]  sessionKeyBA;
}
```

```
//*********************************************************************
// File:         AgentPacket.java
// Name:         LT Roy J. Virden
// Date:         24 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
// Description:
//*********************************************************************

package agentUtil;
import java.io.*;


//---------------------------------------------------------------------
//  AgentPacket class
//---------------------------------------------------------------------
public class AgentPacket implements Serializable {

   public boolean certificateRequestReply = false;
   public byte[]  agentFolderBA;
   public byte[]  agentFolderBASignature;
   public byte[]  signedCertificateBA;


   //--------------------
   // AgentPacket Certification Request or Reply constructor
   //   (for debug)
   //--------------------
   public AgentPacket(boolean r) {

      certificateRequestReply = r;
   }


   //--------------------
   // AgentPacket Certification Request or Reply constructor
   //--------------------
   public AgentPacket(boolean r, byte[] b) {

      signedCertificateBA = b;
   }


   //--------------------
   // AgentPacket AgentFolder constructor
   //--------------------
   public AgentPacket(byte[] b, byte[] s) {

      certificateRequestReply       = false;
      agentFolderBA          = b;
      agentFolderBASignature = s;
   }
}
```

```
//*********************************************************************
// File:          AgentState.java
// Name:          LT Roy J. Virden
// Date:          16 December 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;
import java.io.*;


//---------------------------------------------------------------------
//  AgentState class
//---------------------------------------------------------------------
public class AgentState implements Serializable {

   public boolean initialVisit;
   public String  tempDestination;
   public String  collectedData;
}
```

```
//*********************************************************************
// File:         CodeLoader.java
// Name:         LT Roy J. Virden
// Credit:       This class was authored by Professor Volpano and
//               distributed in his CS3973 Advanced Object-Oriented
//               Programming in Java course.
// Date:         10 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;
import java.util.*;


//---------------------------------------------------------------------
//  CodeLoader class
//---------------------------------------------------------------------
public class CodeLoader {

        public AgentCodeInterface getActive(byte[] b) throws

                InstantiationException,
                IllegalAccessException,
                ClassCastException
        {

                ProgramLoader loader = new ProgramLoader();

                Class classOf = loader.defClass(b, 0, b.length);

                return (AgentCodeInterface)classOf.newInstance();

        }
}


//---------------------------------------------------------------------
//  ProgramLoader class
//---------------------------------------------------------------------
class ProgramLoader extends ClassLoader {

        private Hashtable Classes = new Hashtable();

        // need defClass since defineClass is protected and hence
        // inaccessible to CodeLoader which is in package active

        public Class defClass (byte[] b, int off, int len) {
                return this.defineClass(b, off, len);
        }

        public Class loadClass(String name, boolean resolve)
                throws ClassNotFoundException
        {
           try {
                Class newClass = (Class)Classes.get(name);
```

```
            if (newClass == null) { // not yet loaded

               newClass = findSystemClass(name);
               if (newClass != null)
                    return newClass;

               // class not found -- need to load it
               newClass = Class.forName(name);
               Classes.put(name, newClass);
            }
            return newClass;

         } catch(ClassNotFoundException e) {
            throw new ClassNotFoundException(e.toString());
         }
      }
}
```

```
//*********************************************************************
// File:         CPUScheduler.java
// Name:         LT Roy J. Virden
// Credit:       This class is adapted from a scheduler in Java Threads
//               authored by Scott Oaks and Henry Wong (Oaks, 1997).
// Date:         10 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;

// class CPUScheduler is a round-robin thread scheduler
public class CPUScheduler extends Thread {

   private int timeslice;              // # of millis thread should run
   private Queue threadQueue;          // all the threads to be run
   private static boolean initialized = false;

   // create a scheduler with timeslice t
   public CPUScheduler(int t) {

      if (isInitialized())
         throw new SecurityException("Already initialized");
      threadQueue = new Queue();
      timeslice   = t;
      setPriority(6);
      setDaemon(true);
   }

   // test for existing scheduler
   private synchronized static boolean isInitialized() {

      if (initialized)
         return true;

      initialized = true;
      return false;
   }


   // add a thread to the scheduler's thread queue
   public synchronized void addThread(Thread t) {

      t.setPriority(2);
      threadQueue = threadQueue.insertQ(t);
   }


   // schedules threadQueue
   public void run() {

      Thread current = null;

      while (true) {

         synchronized (this) {
```

```
            while (threadQueue.isemptyQ()) {
               try {
                  this.wait();
               } catch (InterruptedException ie) { }
            }

            current     = (Thread) threadQueue.frontQ();
            threadQueue = threadQueue.leaveQ();
         }

         try {
            current.setPriority(4);
         } catch (Exception e) { continue; };  // don't reinsert thread

         try {
            Thread.sleep(timeslice);
         } catch (InterruptedException ie) { };

         try {
            current.setPriority(2);
         } catch (Exception e) { continue; };  // don't reinsert thread

         synchronized (this) {
            threadQueue = threadQueue.insertQ(current);
         }

      }
   }
}
```

```
//*********************************************************************
// File:          PrincipalHost.java
// Name:          LT Roy J. Virden
// Date:          25 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;

import java.net.*;
import java.io.*;
import java.util.*;
import java.security.*;


//---------------------------------------------------------------------
//  PrincipalHost class
//---------------------------------------------------------------------
public class PrincipalHost implements Principal, Serializable {

   String name;


   //--------------------
   // constructor
   //--------------------
   public PrincipalHost(String nameIn) {
      this.name = nameIn;
   }


   //--------------------
   // equals method compares string objects
   //--------------------
   public boolean equals(Object o) {
      String nameIn = (String)(o);
      boolean match = this.getName().equals(nameIn);
      return match;
   }


   //--------------------
   // getName method returns a String
   //--------------------
   public String getName() {
      return this.name;
   }


   //--------------------
   // hashCode not implemented
   //--------------------
   public int hashCode() {
      return 0;
   }
```

```
    //--------------------
    // toString method returns a String
    //--------------------
    public String toString() {
       return this.getName();
    }
}
```

```
//*********************************************************************
// File:         Queue.java
// Name:         LT Roy J. Virden
// Credit:       This class was authored by Professor Volpano and
//               distributed in his CS3973 Advanced Object-Oriented
//               Programming in Java course.
// Date:         10 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
//*********************************************************************

package agentUtil;
import java.io.*;

public class Queue {

   private Object data;
   private Queue  next;


   // constructor for an empty Queue
   public Queue() {
      this.data = null;
      this.next = this;
   }


   // inserts object at back of queue
   public synchronized Queue insertQ(Object object) {

      Queue newNode = new Queue();
      this.data     = object;
      newNode.next  = this.next;
      this.next     = newNode;

      return newNode;
    }


   // returns object at front of queue
   public synchronized Object frontQ() {

      return this.next.data;
   }


   // delete object at front of queue
   public synchronized Queue leaveQ() {

      if (isemptyQ()) { }
      else {
         this.next = this.next.next;
      }
      return this;
   }
```

```
    // returns true if queue is empty
    public synchronized boolean isemptyQ() {

        if (this.next == this) {
            return true;
        }
        else {
            return false;
        }
    }


    // provides output of Queue
    public synchronized String toString(){

        String s = new String("\n");

        for (Queue q = this.next; q != this; q = q.next)
            s = s + q.data.toString() + "\n";

        return s + "---";
    }
}
```

```
//*********************************************************************
// File:         SignedCertificate.java
// Name:         LT Roy J. Virden
// Date:         24 November 1997
// Advisor:      Professor Dennis Volpano
// System:       Solaris V2.5  JDK1.1.2
// Description:
//*********************************************************************

package agentUtil;
import java.io.*;


//---------------------------------------------------------------------
//  SignedCertificate class
//---------------------------------------------------------------------
public class SignedCertificate implements Serializable {

   public byte[]  certificateBA;
   public byte[]  certificateBASignature;
}
```

4. MISCELLANEOUS

```
//*********************************************************************
// File:          data (header not included in data file)
// Name:          LT Roy J. Virden
// Date:          29 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************

Two F-4 aircraft departed Airfield Alpha at 1003Z
```

```
//*********************************************************************
// File:          dest (header not included in dest file)
// Name:          LT Roy J. Virden
// Date:          29 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************

ro140206.cc.nps.navy.mil
```

```
//*********************************************************************
// File:          typescript (this header not included in printout)
// Name:          LT Roy J. Virden
// Date:          29 November 1997
// Advisor:       Professor Dennis Volpano
// System:        Solaris V2.5  JDK1.1.2
//*********************************************************************


<114 ropub3(Solaris) /test/prototype3> java StartAgent
uploading agentCode:
Destination Authenticated
agentPacket forwarded to ro140203.cc.nps.navy.mil
StartAgent finished

APS: ap.certificateRequestReply = true
APS: agentFolder verified!
agentCode:  I'm at Home (ro140203.cc.nps.navy.mil)
agentCode:  sleeping 2 seconds. . . .
agentCode:  forwarding agent to TIHQ (ro140204)
Destination Authenticated
agentPacket forwarded to ro140204.cc.nps.navy.mil

APS: ap.certificateRequestReply = true
APS: agentFolder verified!
agentCode:  I have returned Home (ro140203.cc.nps.navy.mil)
agentCode:  sleeping 2 seconds. . . .
agentCode:  Here is the data you requested:
agentCode:  Two F-4 aircraft departed Airfield Alpha at 1003Z

APS: ap.certificateRequestReply = true
APS: agentFolder verified!
agentCode:  I have returned Home (ro140203.cc.nps.navy.mil)
agentCode:  sleeping 2 seconds. . . .
agentCode:  Here is the data you requested:
agentCode:  Three F-4 aircraft departed Airfield Alpha at 1015Z
```

LIST OF REFERENCES

Black, D. L., Kahn, C., *Mobile Agents and Network Survivability,* position paper for the Information Survivability Workshop – ISW'97, San Diego, California, February 12-13, 1997.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G., *Itinerant Agents for Mobile Computing*, IEEE Personal Communications Magazine, 2(5):34-49, October 1995. http://www.research.ibm.com/massive

Edmiston, M., Gregg, D., Wirth, D., *Decision Support for Reconnaissance Using Intelligent Software*, Masters Thesis, Naval Postgraduate School, in preparation.

Farley, S. R., *Mobile Agent System Architecture*, Java Report, p. 39, May 1997.

Farmer, W. M., Guttman, J. D., and Swarup, V., *Security for mobile agents: Issues and requirements*. In Proceedings of the 19th National Information Systems Security Conference, pages 591-597, Baltimore, Md., October 1996

Harrison, C. G., Chess, D. M., Kershenbaum, A., *Mobile Agents: Are they a good idea?*, Research Report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, March 28, 1995.

Kalakota, R., Whinston, A. B., *Frontiers of Electronic Commerce*, p. 596, Addison-Wesley Publishing Co., 1996.

Minar, N., *Computational Media for Mobile Agents*, paper submitted for MAS737: Software Agents Seminar, December 31, 1996.

Minsky, M., *The Society of Mind*, Simon and Schuster, Inc., 1985.

Muller, J. P., *The Design of Intelligent Agents: A Layered Approach*, Springer-Verlag, 1996.

Oaks, S., Wong, H., *Java Threads*, O'Reilly and Associates, Inc., January 1997.

Russell, D., Gangemi, Sr., G.T., *Computer Security Basics*, O'Reilly and Associates, Inc., July 1991.

Stallings, W., *Data and Computer Communications*, p. 687, Prentice Hall, 1997.

Volpano, D. M., unpublished manuscript, 1997

Yee, B., *A Sanctuary for Mobile Agents*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

BIBLIOGRAPHY

Alvarez, C., *Intelligent agents help to minimize bandwidth use by browsing offline*, LAN Times, January 20, 1997.

An Introduction to Safety and Security in Telescript, Northeast Parallel Architectures Center at Syracuse University, Syracuse NY available at http://king.syr.edu:2006/Misc/IWT/Technologies/Telescript/SafeTS.html

Black, D. L., Kahn, C., *Mobile Agents and Network Survivability,* position paper for the Information Survivability Workshop – ISW'97, San Diego, California, February 12-13, 1997.

Burrows, M., Abadi, M., Needham, R., *A Logic of Authentication*, ACM Trans. Computer Systems, vol. 8, pp. 18-36, February, 1990.

Cardelli, L., *Mobile computations*, In Mobile Object Systems: Towards the Programmable Internet, pages 3-6. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G., *Itinerant Agents for Mobile Computing*, IEEE Personal Communications Magazine, 2(5):34-49, October 1995. http://www.research.ibm.com/massive

Crowston, K., *Market-Enabling Internet Agents*, Syracuse University.

Dean, D., Felten, E., *Secure Mobile Code: Where do we go from here?*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Denning, D. E., *Cryptography and Data Security*, Reading, Massachusetts: Addision-Wesley Publishing Company,1983.

Dyer, D., *Agent-Based Systems Concept*, DARPA Agent Based Systems Program presentation, July 16, 1997. http://abs.wwwhome.com

Edmiston, M., Gregg, D., Wirth, D., *Decision Support for Reconnaissance Using Intelligent Software*, Masters Thesis, Naval Postgraduate School, in preparation.

Farley, S. R., Mobile Agent System Architecture, Java Report, p. 39, May 1997.

Farmer, W. M., Guttman, J. D., and Swarup, Vipin, *Security for mobile agents: Issues and requirements*. In Proceedings of the 19th National Information Systems Security Conference, pages 591-597, Baltimore, Md., October 1996.

Feigenbaum, J., Lee, P., *Trust Management and Proof-Carrying Code in Secure Mobile-Code Applications*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Focardi, R., Gorrieri, R., *Non Interference: Past, Present and Future*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Fournet, C., *Security within a Calculus of Mobile Agents?*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Franklin, M. K., Reiter, M. K., *The Design and Implementation of a Secure Auction Service*, In Proceedings of IEEE Symposium on Security and Privacy, pp2-14, Oakland, California, May 8-10, 1995.

Gong, L., *New Security Architectural Directions for Java (Extended Abstract)*, In Proceedings of IEEE COMPCON, pp.97-102, San Jose, California, February 1997.

Gong, L., *Surviveable Mobile Code is Hard to Build*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Gordon, A. D., *Nominal Calculi for Security and Mobility*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Gray, R. S., *Agent Tcl: A flexible and secure mobile-agent system*, In Proceedings of Fourth Annual Usenix Tcl/Tk Workshop, pp. 9-23, 1996.

Gunter, C., Homeier, P., Nettles, S., *Infrastructure for Proof-Referencing Code*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Harrison, C. G., Chess, D. M., Kershenbaum, A., *Mobile Agents: Are they a good idea?*, Research Report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, March 28, 1995.

Heintze, N., Riecke, J., *The SLam Calculus: Programming with Security and Integrity*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

IBM Corporation, *Things that Go Bump in the Net*, Web page at http://www.research.ibm.com/massive, 1995.

Jan V., *Secure object spaces*. In Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, pages 41-48, Linz, Austria, July 1996.

Kalakota, R., Whinston, A. B., *Frontiers of Electronic Commerce*, pp 595-627, Addison-Wesley Pub Co, January 1996.

Kalakota, R., Stallaert, J., Whinston, A. B., *Mobile Agents and Mobile Workers*, Proceedings of the 29th Annual Hawaii International Conference on System Sciences, 1996.

Kato, K., Toumura K., Matsubara K., Aikawa S., Yoshida J., Kono K., Taura K., and Sekiguchi T., *Protected and secure mobile object computing in PLANET*. In Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, Linz, Austria, July 1996.

Lampson, B., Abadi, M., Burrows, M., Wobber, E., *Authentication in Distributed Systems: Theory and Practice*, Proceedings of the 13th ACM Symposium on Operating Systems Principles, 1991.

Lange, D. B., Chang, D. T., *Programming Mobile Agents in Java*, a white paper, IBM Corporation, September 9, 1996.

Lee, P., Necula, G., *Research on Proof-Carrying Code for Mobile-Code Security*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Meadows, C., *Detecting Attacks on Mobile Agents*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Meseguer, J., Talcott, C., *Rewriting Logic and Secure Mobility*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Minar, N., *Computational Media for Mobile Agents*, paper submitted for MAS737: Software Agents Seminar, December 31, 1996.

Minsky, M., *The Society of Mind*, Simon and Schuster, Inc., 1985.

Minsky, Y., Rensse, R., Schneider, F. B., *Crytpographic Support for Fault-Tolerant Distributed Computing*, Department of Computer Science, Cornell University, Ithaca, NN, July 5, 1996.

Muller, J. P., *The Design of Intelligent Agents: A Layered Approach*, Springer-Verlag, 1996.

Oaks, S., Wong, H., *Java Threads*, O'Reilly and Associates, Inc., January 1997.

Russell, D., Gangemi, Sr., G.T., *Computer Security Basics*, O'Reilly and Associates, Inc., July 1991.

Sommers, B., *Agents: Not just for Bond anymore*, Javaworld, March 15, 1997.

Swarup, V., *Trust Appraisal and Secure Routing of Mobile Agents*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

Thirunavukkarasu, C., Finin, T., Mayfield, J., *Secret Agents - A Security Architecture for the KQML Agent Communications Language*, in CIKM workshop on Intelligent Information Agents, Baltimore, December 1995.

Venners, B., *Under the Hood: The architecture of aglets, Javaworld*, March 17, 1997.

Volpano, D. M., unpublished manuscript, 1997.

Wetherall, D., *Safety Mechanisms for Mobile Code*, Area Examination Paper, Telemedia Networks and Systems Group, Laboratory for Computer Science, Massachusetts Institute of Technology, November 1995.

Yee, B., *A Sanctuary for Mobile Agents*, position paper from DARPA Workshop on Foundations for Secure Mobile Code, March 26-28, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Commanding Officer 1
   (Attn: Code 30, CDR Zellmann)
   Naval Information Warfare Activity
   9800 Savage Rd.
   Ft Meade, MD 20755-6000

4. National Security Agency 1
   Suite 6704 ATTN: Steve LaFountain, C4
   Fort George G. Meade, MD 20755-6000

5. Commander, Naval Security Group Command 1
   Naval Security Group Headquarters
   9800 Savage Road, Suite 6585
   Fort George G. Meade, MD 20755-6585
   ATTN: N6/Mr. James H. Shearer

6. ECJ6-NP 1
   HQUSEUCOM
   Unit 30400 Box 1000
   APO, AE 09128

7. Dr. Dan Boger 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

8. Dr. Dennis Volpano 1
   Code CS/Vo
   Naval Postgraduate School
   Monterey, CA 93943

9. Major Nelson Ludlow 1
Air Force Radar Evaluation Squadron
7976 Aspen Avenue
Hill AFB, UT 84056-5846

10. Commander Gus Lott 1
Code EC/Lt
Naval Postgraduate School
Monterey, CA 93943-5121

11. Dr. Cynthia Irvine 1
Code CS/Ic
Naval Postgraduate School
Monterey, CA 93943

12. Dr. Deborah Hensgen 1
Code CS/Hd
Naval Postgraduate School
Monterey, CA 93943

13. Dr. Don Brutzman 1
Code UW/Br
Naval Postgraduate School
Monterey, CA 93943